XML

定义

XML，即 extensible Markup Language ， 是一种数据标记语言 & 传输格式

作用

对数据进行标记（结构化数据）、存储 & 传输

特性

1. 灵活性: 可自定义标签，文档结构
2. 自我描叙性
    - XML文档即 一个纯文本文件，代码结构清晰，适合人类阅读
    - 有文本处理能力的软件都可以处理XML
3. 可扩展性: 可在不中断解析，应用程序的情况下进行扩展
4. 可跨平台数据传输: 可以不兼容的系统间交换数据，降低了复杂性
5. 数据共享: XML 以纯文本进行存储，独立于软硬件和应用程序的数据存储方式，使得不同的系统都能访问XML

语法

- 元素要关闭标签
- 对大小写敏感
- 必须要有根元素(父元素)
- 属性值必须加引号
- XML元素命名规则
    - 不能以数字或标点符号开头
    - 不能包含空格
    - 不能以xml开头
- CDATA
  不被解析器解析的文本数据，所有xml文档都会被解析器解析（cdata区段除外）

- PCDATA
  被解析的字符数据

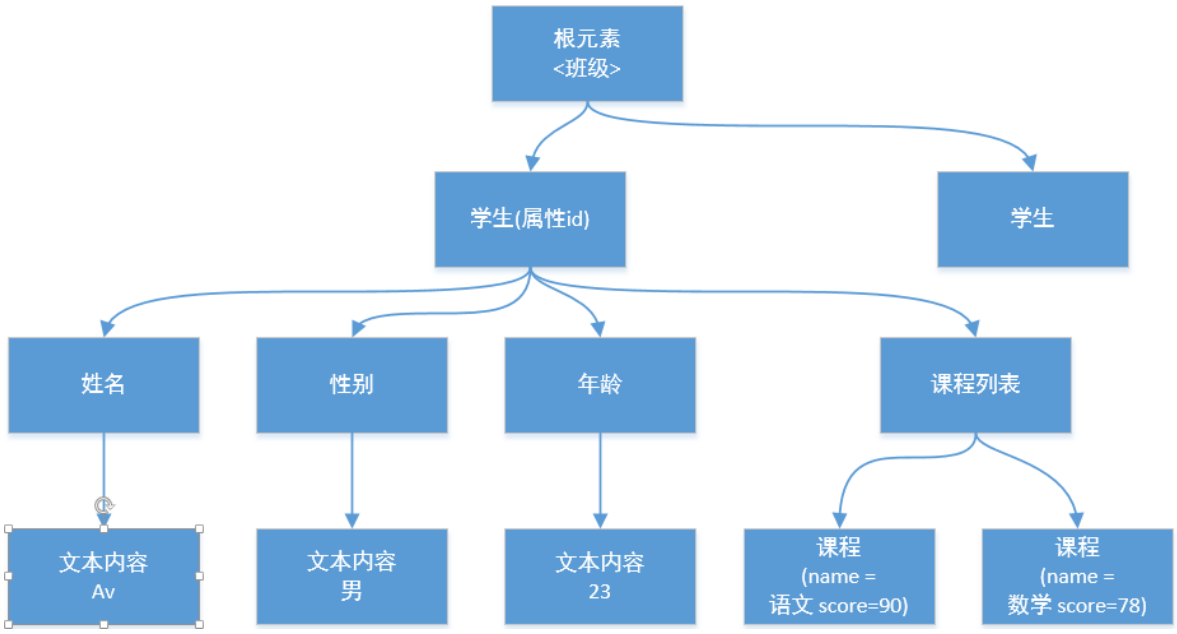
XML树形结构

XML文档中的元素会形成一种树结构，从根部开始，然后拓展到每个树叶（节点）,下面将以实例说明XML的树结构。

```
<?xml version="1.0" encoding="utf-8"?>
<classes><!--根节点 -->
    <student id="0">
```

```
        <name>Av</name>
        <age>23</age>
        <sax>男</sax>
        <Courses>
            <course name="语文" score="90"/>
            <course name="数学" score="78"/>
        </Courses>
    </student>
    <student id="1">
        <name>Lance</name>
        <age>22</age>
        <sax>男</sax>
        <Courses>
            <course name="语文" score="59"/>
            <course name="数学" score="38"/>
        </Courses>
    </student>
</classes>
```

树形结构



XML节点解释

XML文件是由节点构成的。它的第一个节点为“根节点”。一个XML文件必须有且只能有一个根节点，其他节点都必须是它的子节点,每个子节点又可以有自己的子节点。

解析方式

解析XML，即从XML中提取有用的信息
XML的解析方式主要分为2大类：

解析方式	原理	类型
基于文档驱动	在解析XML文档前，需先将整个XML文档加载到内存中	DOM方式
基于事件驱动	根据不同需求事件(检索，修改，删除等)去执行不同解析操作(不需要把整个XML 文档加载到内存中)	SAX方式，PULL方式

DOM方式

Document Object Model，即 文件对象模型，是 一种 基于树形结构节点 & 文档驱动 的XML解析方法，它定义了访问 & 操作xml文档元素的方法和接口

DOM解析原理

- 核心思想
  基于文档驱动，在解析XML文档前，先将整个XML文档存储到内存中，然后再解析
- 解析过程
  1. 解析器读入整个XML文档到内存中
  2. 解析全部文件，并将文件分为独立的元素，属性等，以树结构的形式在内存中表示XML文件
  3. 然后通过DOM API去遍历XML树，根据需要搜索数据/修改文档
- 具体解析步骤
  1. 获取DOM解析器工厂实例(DocumentBuilderFactory.newInstance())
  2. 获取DOM解析器对象,调用解析器工厂实例类的newDocumentBuilder()
  3. 最后获取代表整个文档的Document对象

具体解析实例

```
public void domTest(Context context) {
        try {
            DocumentBuilderFactory dbf =
DocumentBuilderFactory.newInstance();
            DocumentBuilder db = dbf.newDocumentBuilder();
            Document document =
db.parse(context.getResources().openRawResource(R.raw.students));
            //通过Document对象的getElementsByTagName()返根节点的一个list集合
            NodeList studentList = document.getElementsByTagName("student");
            for (int i = 0; i < studentList.getLength(); i++) {
                Student student = new Student();
                //循环遍历获取每一个student
                Node studentNode = studentList.item(i);
                if (((Element) studentNode).hasAttribute("id")) {
                    student.setId(Integer.parseInt(((Element)
studentNode).getAttribute("id")));
                }
                //解析student节点的子节点
                NodeList childList = studentNode.getChildNodes();
                for (int t = 0; t < childList.getLength(); t++) {
                    //区分出text类型的node以及element类型的node
                    if (childList.item(t).getNodeType() ==
Node.ELEMENT_NODE) {
                        if
(childList.item(t).getNodeName().equalsIgnoreCase("Courses")) {
                            NodeList courses =
childList.item(t).getChildNodes();
                            for (int j = 0; j < courses.getLength(); j++) {
                                Node courseNode = courses.item(j);
                                if (courseNode.getNodeType() !=
Node.ELEMENT_NODE) {
                                    continue;
                                }
```

```
                                NamedNodeMap namedNodeMap =
courseNode.getAttributes();
                                Course course = new Course();
                                student.addCourse(course);
                                for (int k = 0; k <
namedNodeMap.getLength(); k++) {
                                    Node courseAttr = namedNodeMap.item(k);
                                    if
(courseAttr.getNodeName().equals("name")) {

 course.setName(courseAttr.getNodeValue());
                                    } else if
(courseAttr.getNodeName().equals("score")) {

 course.setScore(Float.parseFloat(courseAttr.getNodeValue()));
                                    }
                                }
                            }
                        } else {
                            Node child = childList.item(t);
                            if (child.getNodeName().equals("name")) {
                                student.setName(child.getTextContent());
                            } else if (child.getNodeName().equals("age")) {

 student.setAge(Integer.parseInt(child.getTextContent()));
                            } else if (child.getNodeName().equals("sax")) {
                                student.setSax(child.getTextContent());
                            }
                        }
                    }
                }
                Log.i("Zero", "解析完毕: " + student);
            }
        } catch (ParserConfigurationException | SAXException | IOException
e) {
            Log.e("Zero", e.getMessage());
        }
    }
```

特点及应用场景

- 优点
  - 操作整个XML文档的效率高
  - 可随时，多次访问已解析的文档
- 缺点
  - 耗内存，时间
    应用场景
  - 适合XML文档较小，需频繁操作 解析文档，多次访问文档的情况
  - 对于移动端，内存资源非常宝贵，使用时需权衡利弊

SAX方式

即 Simple API for XML，一种 基于事件流驱动、通过接口方法解析 的XML解析方法

SAX解析原理

- 核心思想
  基于事件流驱动，根据不同需求事件(检索，修改，删除等)去执行不同解析操作，不需要把整个XML 文档加载到内存中
- 解析过程
  1. 按顺序扫描XML文档
  2. 当扫描到(Document)文档的开始/结束标签，(Element)节点元素的开始/结束标签时，直接调用对应的方法，将状态信息以参数的方式传递到方法中
  3. 然后根据状态信息去执行相关的自定义操作
     具体的操作
  4. 自定义Handler处理类，继承自DefaultHandler类
  5. 重写5个核心回调方法

```
startDocument()
startElement()
characters()
endElement()
endDocument()
```

具体解析实例

```
  public void saxTest(Context context) throws Exception {
        SAXParserFactory spf = SAXParserFactory.newInstance();
        SAXParser sp = spf.newSAXParser();
        sp.parse(context.getResources().openRawResource(R.raw.students), new
DefaultHandler() {

            String currentTag = null;
            Student student = null;

            /**
             * 文档解析开始时被调用
             * @throws SAXException
             */
            @Override
            public void startDocument() throws SAXException {
                super.startDocument();
            }

            /**
             * 文档解析结束时被调用
             * @throws SAXException
             */
            @Override
            public void endDocument() throws SAXException {
                super.endDocument();
            }

            /**
             *
             * @param uri 命名空间
             * @param localName 不带命名空间前缀的标签名
             * @param qName 带命名空间的标签名
             * @param attributes 标签的属性集合 <student id="0"></student>
             * @throws SAXException
```

```
             */
            @Override
            public void startElement(String uri, String localName, String
qName, Attributes attributes) throws SAXException {
                super.startElement(uri, localName, qName, attributes);
                currentTag = localName;
                if ("student".equals(currentTag)) {
                    student = new Student();

 student.setId(Integer.parseInt(attributes.getValue("id")));
                }
                if ("course".equals(currentTag)) {
                    if (student != null) {
                        Course course = new Course();
                        course.setName(attributes.getValue("name"));

 course.setScore(Float.parseFloat(attributes.getValue("score")));
                        student.addCourse(course);
                    }
                }

            }

            /**
             *  解析到结束标签时被调用 '/>'
             * @param uri
             * @param localName
             * @param qName
             * @throws SAXException
             */
            @Override
            public void endElement(String uri, String localName, String
qName) throws SAXException {
                super.endElement(uri, localName, qName);
                if ("student".equals(localName)) {
                    Log.i(TAG, "endElement: student: " + student);
                }
            }

            /**
             *
             * @param ch 内容
             * @param start 起始位置
             * @param length 长度
             * @throws SAXException
             */
            @Override
            public void characters(char[] ch, int start, int length) throws
SAXException {
                super.characters(ch, start, length);
                String str =  new String(ch, start, length).trim();
                if (TextUtils.isEmpty(str))
                    return;
                if ("name".equals(currentTag) && student != null) {
                    student.setName(str);
                }
                if ("age".equals(currentTag)&&  student != null) {
                    student.setAge(Integer.parseInt(str));
```

```
                }
                if ("sax".equals(currentTag) && student != null) {
                    student.setSax(str);
                }

            }
        });
    }
```

特点及应用场景

- 优点
  - 解析效率高
  - 内存占用少
- 缺点
  - 解析方法复杂，API接口方法复杂，代码量大
  - 可扩展性差，无法修改XML树内容结构
    应用场景
  - 适合XML文档大，解析性能要求高，不需修改 多次访问解析的情况

PULL方式

一种 基于事件流驱动 的XML解析方法，是Android系统特有的解析方式

PULL解析原理

基于事件流驱动，根据不同需求事件(检索，修改，删除等)去执行不同解析操作，不需要把整个XML 文档加载到内存中

- 解析过程
  1. 首先按顺序扫描XML文档
  2. 解析器提供文档的开始/结束(START_DOCUMENT,END_DOCUMENT)，元素的开始/结束(START_TAG,END_TAG)
  3. 当某个元素开始时，通过调用parser.nextText()从XML文档中提取所有字符数据

具体解析实例

```
 public void pullTest(Context context) throws Exception {
        XmlPullParser parser = XmlPullParserFactory.newInstance().newPullParser();

 parser.setInput(context.getResources().openRawResource(R.raw.students), "utf-8");//设置数据源编码
        int eventCode = parser.getEventType();//获取事件类型
        Student student = null;
        while (eventCode != XmlPullParser.END_DOCUMENT) {
            switch (eventCode) {
                case XmlPullParser.START_DOCUMENT://开始读取XML文档
                    break;
                case XmlPullParser.START_TAG://开始读取标签
                    String name = parser.getName();
                    if ("student".equals(name)) {
                        student = new Student();
```

```
14
                    student.setId(Integer.parseInt(parser.getAttributeValue(null, "id")));
15                  }
16                  if ("name".equals(name) && student != null) {
17                      student.setName(parser.nextText());
18                  }
19                  if ("age".equals(name) && student != null) {
20
                    student.setAge(Integer.parseInt(parser.nextText().trim()));
21                  }
22                  if ("sax".equals(name) && student != null) {
23                      student.setSax(parser.nextText());
24                  }
25                  if ("course".equals(name) && student != null) {
26                      Course course = new Course();
27                      course.setName(parser.getAttributeValue(null,
                    "name"));
28
                    course.setScore(Float.parseFloat(parser.getAttributeValue(null, "score")));
29                      student.addCourse(course);
30                  }
31                  break;
32              case XmlPullParser.END_TAG://结束原始事件
33                  if ("student".equals(parser.getName())) {
34                      Log.i(TAG, "pullTest: student: " + student);
35                  }
36                  break;
37
38          }
39          eventCode = parser.next();
40      }
41
42    }
```

特点及应用场景

- 优点
  - 解析效率高
  - 内存占用少
  - 灵活性高 可控制事件处理结束的时机(与SAX最大的区别)
  - 使用比SAX方式简单
- 缺点
  - 可扩展性差，无法修改XML树内容结构
    应用场景
  - 适合XML文档大，解析性能要求高，不需修改 多次访问解析的情况
  - Pull使用比SAX更加简单，在Android中推荐使用Pull方式